Keyword Aware Influential Community Search in Large Attributed Graphs

Md. Saiful Islam
BUET, Bangladesh
saifulislam@cse.buet.ac.bd

Mohammed Eunus Ali
BUET, Bangladesh
eunus@cse.buet.ac.bd

Yong-Bin Kang
Swinburne University of
Technology, Australia
ykang@swin.edu.au

Timos Sellis
Swinburne University of
Technology, Australia
tsellis@swin.edu.au

Farhana M. Choudhury
Melbourne University,
Australia
fchoudhury@unimelb.edu.au

ABSTRACT

We introduce a novel keyword-aware influential community query ($KICQ$) that finds *the most influential communities* from an attributed graph, where an influential community is defined as a closely connected group of vertices having some dominance over other groups of vertices with the expertise (a set of keywords) matching with the query terms (words or phrases). We first design the $KICQ$ that facilitates users to issue an influential CS query intuitively by using a set of query terms, and predicates (AND or OR). In this context, we propose a novel word-embedding based similarity model that enables *semantic community search*, which substantially alleviates the limitations of *exact keyword* based community search. Next, we propose a new influence measure for a community that considers both the cohesiveness and influence of the community and eliminates the need for specifying values of internal parameters of a network. Finally, we propose two efficient algorithms for searching influential communities in large attributed graphs. We present detailed experiments and a case study to demonstrate the effectiveness and efficiency of the proposed approaches.

1. INTRODUCTION

Communities serve as a basic structure for understanding the organization of many real-world networks or graphs. These networks include academic networks like DBLP, social networks like Facebook or Twitter, biological networks like protein-protein interactions, and many more. Finding
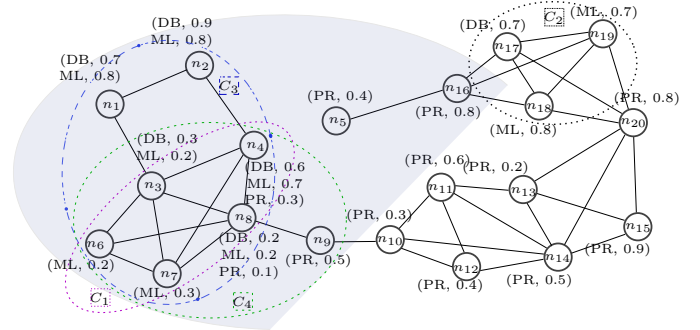
Figure 1: An attributed author-author graph, where each vertex has an associated list of attributes (keywords) and influences denoting her expertise. Different types cf communities are marked as $C_1 - C_4$.

communities from such large graphs has received significant attention in recent years due to its diverse practical applications that include event organization [38], friend recommendation [33], and e-commerce advertisement [22]. Traditionally, community search (CS) on a large graph involves finding a community around a given query vertex that satisfies *query parameters* like *connectivity* and *cohesiveness constraints* [13, 12, 18, 38]. For example, by using such techniques, one can find a community from the DBLP network for an author as a query, where the community should be a connected subgraph and each member should be connected to at least two other members in the community. More recent research works [27, 25] have focused on *finding influential communities* from a graph. The common goal of finding influential communities is to find a closely connected group of users (vertices) who have some dominance over other users in the graph in a particular domain.

In this paper, we consider large attributed graphs where vertices (e.g., authors) are augmented with attributes (e.g., keywords) and propose a novel and efficient solution for finding influential communities that address the following gaps in the previous works:

First, traditional CS works on an attributed graph require an input query vertex, and then find a group of neighboring vertices whose keywords have high similarity with the query vertex keywords. The resultant communities satisfy the required structural constraints [13, 18] (e.g., C_1 in Figure 1 with n_3 as the query vertex, and parameter $k = 4$

where k-truss is the structural constraint). A major limitation of such CS techniques is that the user needs to define the query vertex and the structural properties of the community explicitly, which might not be possible or suitable in many application domains. A couple of recent studies [46, 4] tried to address these limitations by finding cohesive (i.e., k-core or triangle density) communities having close similarity with query keywords. However, they do not consider the influence of individuals in different keywords (e.g., C_1 in Figure 1 is highly cohesive in terms of structure and keyword, but two highly influential vertices n_1, n_2 are ignored since influence is not considered) and also do not support flexible conjoining (using AND or OR predicates) of query keywords.

Second, existing works on influential community search only work on non-attributed graphs and also require specific values of structural parameters. For example, [27] requires users to mention the value of k while finding k-core based communities; similarly, [25] requires the values of the minimum number of vertices m in a community and the maximum distance p between any two vertices while finding an mp-clique based community. Although such parameters allow high customization in search, we argue that the choice of these parameters highly depends on the internal structure of the graph in practice. For example, if k is set to a high value (e.g., 4) in a small graph (Figure 1), no community is returned by [27] because there is no 4-core in this graph; and if k is low (e.g., 2), the community returned (e.g., C_2 in Figure 1) does not have high cohesiveness. Similarly, given a query vertex, [25] only returns the desired community under specific constraints of parameter values (e.g., C_4 in Figure 1 with parameters $m \leq 6$ and $p = 2$), which is impractical for an external user. Thus flexibility in fixing parameters while searching for desired communities is crucial.

Third, existing approaches to quantifying a community in terms of influence do not consider a comprehensive set of parameters that can affect the strength of a community. For example, the influence of a community is defined as the minimum influence among all members in [27]; thus, a member with low influence can severely affect the influence of a community. We argue that an influence measure that considers both cohesiveness, the influence of individuals, and the size of the community should be considered while ranking communities, as all these factors contribute to the overall ranking of a community.

To fill the above research gaps and to support a new set of applications, we present a novel parameter free influential CS query, namely Top-r Keyword-aware Influential Community Query ($KICQ$). To illustrate, let us consider an attributed graph of researchers, as shown in Figure 1. Here, vertices $n_1 - n_{20}$ are authors who published papers in field of studies relevant to "Machine Learning (ML)", "Database (DB)", and "Pattern Recognition (PR)". An aspirant Ph.D. student may be interested in finding the most influential community who are working in "ML" or "DB." The $KICQ$ returns the community C_3 as shown in Figure 1 as the most influential community (see Section 5 for the influential metric) since the members of the community have influence in either "ML" or "DB", the community is dense and also contains highly influential members.

The $KICQ$ can be useful for retrieving influential communities in other social networks, where a user is connected with her friends/followers, and her preferences are extracted from her posts/likes/shared contents. From such a network, one event organizer may want to find the most influential communities who have interests in "music or movies". Similarly, a tour operator may want to identify the groups who have "travel" as their interests as potential customers.

A major challenge in realizing such a query (i.e., $KICQ$) comes from the fact that communities and their influences need to be computed and compared on the fly based on the set of keywords in the query, and thus existing precomputation based approaches are not suitable for our purpose. Our major contributions are the following:

First, we design $KICQ$ in such a way that enables users to issue an influential community search query intuitively by merely using a set of query terms (words or phrases), and predicates (AND or OR) (addressing the first limitation). In this context, we propose a novel word-embedding based keyword similarity model that enables *semantic community search*, which substantially alleviates the limitations of *exact keyword* based community search. For example, a user may use "song" instead of "music," where exact search fails to retrieve the relevant communities if the attributed graph contains "music" as a keyword. *This approach is of independent interest in enhancing any knowledge graph with semantically meaningful sets of words.* (Section 4)

Second, we propose a new influence measure for a community that considers both the cohesiveness and influence of the community and eliminates the need for specifying values of internal parameters of a network (addressing the second limitation). The influence measure also captures the influence of individual members in a better intuitive sense rather than the influence of the community being dominated by the minimum influence of a member (addressing the third limitation). We demonstrate the effectiveness of the proposed measure in a case study. (Section 5)

Third, we propose two efficient algorithms for searching influential communities in a large, attributed graph. The basis of the first algorithm is pruning the communities that cannot be a part of the answer set based on the computed scores of already explored subgraphs. The second one is a novel tree-based approach, where we augment the tree with influence score bounds for each keyword and prune the unnecessary branches of the tree based on the scores of the explored community. (Section 6)

Fourth, we conduct comprehensive experiments with real datasets to evaluate our proposed algorithms. The experimental results show that our algorithms are highly efficient and effective in retrieving keyword aware influential communities compared to the state-of-the-art influential community search technique. (Section 7)

2. RELATED WORKS

Finding communities from large graphs has been an engaging research direction for a long time. Although the definition of community varies among different studies, cohesive subgraphs like maximal cliques [8], k-core [7], k-truss [40], etc. form the basis of modeling communities. The task of finding communities can be divided into two major classes: community detection (CD), and community search (CS). Recently, Li et al. [27] introduced the notion of *influential community* that has piqued interest from the research community. CS and CD problems are studied on both simple and attributed graphs.

CS Approaches	Simple graph	Attributed graph	
		Keyword	Others
Basic CS	[10, 17, 38, 1, 44, 45, 28]	[13, 18, 9, 46, 21, 4]	[6, 12, 41, 49]
Influential CS	[27, 5, 47, 3, 25]	-	[26]

Table 1: Existing community search works

Team formation, another relevant domain, is the task of finding a subset of available individuals to complete a project which requires a specific set of skills, which can be viewed as a set coverage problem usually with a minimum communication cost objective [23, 24]. These works differ from CS as a team does not need to be a cohesive subgraph.

2.1 Community detection

In CS, communities are defined based on the query, and CS solutions aim to find communities efficiently in an online manner. CD methods usually use global criteria to detect all the communities from an entire graph, where the focus is more on quality (e.g., cohesiveness) than efficiency. Link based analysis was popular in initial studies [16] that did not consider attributes in a graph. Clustering based techniques [48, 35, 20, 43], and topic modeling [29, 36] are used in recent studies on attributed graphs. However, none of the studies enables a user to find specific communities of her interest, which is the main focus of our study.

2.2 Community search

We present different directions of CS studies in Table 1. Most of the basic CS studies on simple graphs [10, 17, 38, 1, 44, 45] find communities containing given query vertices. Li et al. [28] studied persistent communities in a temporal network, in which every edge is associated with a timestamp. Li et al. [27] introduced the notion of influential CS where vertices are assigned an influence score, and the influence of a community is modeled as the minimum influence of the members. Chen et al. [5] and Bi et al. [3] developed faster algorithms to solve the same problem. Zheng et al. [47] studied influential CS in an undirected weighted graph, where the weight of an edge represents the semantic intimacy between two vertices. Li et al. [25] defined a community in terms of kr-clique and designed algorithms to retrieve the most influential community. All of these studies ignore rich information of vertices found in attributed graphs and require several vertices or internal parameters as part of a query, which is very difficult for a user who does not have enough knowledge of the graph.

There are several studies on CS in attributed graphs. Fang et al. [13] proposed the ACQ algorithm to find subgraphs satisfying structural and keyword cohesiveness. Huang et al. [18] also explored attribute driven CS in terms of k-truss. Chen et al. [6] studied CS in an attributed graph where each vertex has a *profile*: a set of keywords arranged in a tree structure. Chobe et al. [9] employed keyword search techniques to facilitate CS in attributed graphs. However, these studies also require a set of vertices and/or internal parameters as part of the query. Few recent works [46, 4, 46] study keyword-based CS that take a set of keywords as input and return a subgraph as the community that has the *best* match with the given set of query keywords. In these works, the cohesiveness of the subgraph is measured differently, i.e., k-core in [46], triangle density in [4], and average proximity in [21]. To decide a single best-matched subgraph, they define functions that consider the presence or absence of keywords and structural cohesiveness in the subgraph.

These studies are different from ours as they only consider the presence or absence of keywords in different vertices of the subgraph and cannot be adapted for the scenario where we need to rank the communities and each vertex has a certain degree of influence in each keyword. There are CS studies on spatial graphs [12, 41, 49] as well, which are of different interest to our problem.

Li et al. [26] studies skyline community search where each vertex is associated with a d-dimensional influence score. However, their study is designed for low values of d (i.e., $d < 5$). With $d = 5$, their algorithms require more than 10^3 seconds in a graph with half a million vertices. This study cannot be extended for an attributed graph where vertices are associated with influence scores in multiple keywords, because there can be millions of keywords (dimension) in such an attributed graph. Also, this approach aims to find communities with a global objective function, and there is no way to search communities of specific interest.

3. PROBLEM DEFINITION AND SYSTEM OVERVIEW

We first define the attributed graph, proposed community model and keyword aware influential community query ($KICQ$), and then present the overview of the system.

Definition 1. (Attributed graph) An attributed graph $G^+(V, E, A)$ is an undirected graph, where V is the set of vertices and E is the set of edges. Each vertex v is associated with a set of tuples of the form $A_v = \{(w_i, s_v(w_i))\}$, where w_i is a keyword and $s_v(w_i) \in [0, 1]$ is the influence score of vertex v in keyword w_i.

We consider the connected components of maximal k-cores as the *influential communities*, where the *influence* is defined as Equation 2 (see Section 5). k is termed as *cohesion factor* in this paper.

Definition 2. (maximal k-core) Let H be a subgraph of G^+, induced by the set of vertices $V_H \subseteq V$. Let the degree of a vertex v in H is denoted by $deg_H(v)$. H is a k-core if $\forall_{v \in V_H} deg_H(v) \geq k$, where k is a non-negative integer. H is a maximal k-core if there is no super k-core in G^+ that contains H.

Now, we define the keyword-aware influential community query, $KICQ$ as follows.

Definition 3. ($KICQ$) Let $G^+(V, E, A)$ be an attributed graph, $q(T, P)$ be a query tuple where $T = \{t_1, t_2, \cdots, t_n\}$ is a set of terms (i.e., words or phrases) and P is a predicate (AND, OR) for conjoining the query terms, k_{min} be the minimum cohesion factor of the resultant community, and r be a positive integer specifying the number of top communities to be returned. Then, $KICQ$ finds r *most influential communities* $H_1, H_2, ..., H_r$ from G^+.

An overview of our system is presented in Figure 2. The system is mainly divided into two phases. First, we construct a keyword-aware attributed graph from a social network corpus that may consist of a combination of structured and/or unstructured (i.e., text) data. In an academic domain, the corpus can be scientific publications of researchers (e.g., authors, titles, abstracts, author-provided keywords,
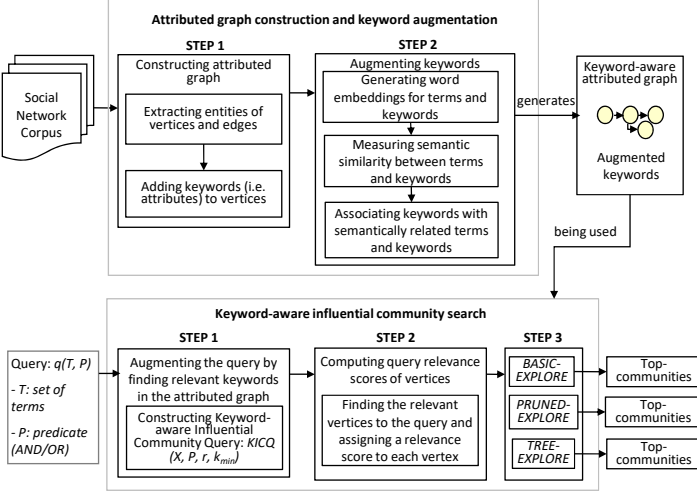
Figure 2: The overview of our system

etc). Second, we focus on searching keyword-aware influential communities using the constructed attributed graph, given a query as a set of terms and predicates.

The distinctive features of the first phase are as follows:

STEP 1 : First, we build an attributed graph from a domain corpus by extracting entities of possible vertices and edges to represent the social network.

STEP 2 : To enable keyword-aware influential community search, we augment keywords with their semantically related terms and keywords. We build *word embedding vectors* [31] as an external knowledge source for associating keywords in the graph with semantically related terms and/or keywords. The semantic relatedness is estimated by exploiting the word embedding vectors. The output of this step is a graph, called *keyword-aware attributed graph*.

Further, the unique features of the second phase of our system can be briefly highlighted below:

STEP 1 : Initially, a query raised by a user is given in the form of a pair $q(T, P)$ consisting of a set of query terms T, and a predicate P. The terms need to match with the keywords in the attributed graph to find meaningful communities. We acknowledge the difficulty faced by the users to put the exact terms while raising a query. For example, it is highly likely that some of the users will input "song" instead of "music". To help the users to easily raise a query, we augment each query term with a semantically meaningful set of keywords. The output of this step is a $KICQ$.

STEP 2 : Given a $KICQ$ query, our objective is to find the vertices relevant to the query and then compute their query relevance scores (Equation 1). Relevance score of vertices are used to compute the scores of potential influential communities. We argue that a measure that rewards both the cohesiveness of the community and high influence of the members, and does not require user input of any internal parameters (e.g., k in k-core) is more preferable than the existing influence measures. We propose a linear weighted summation of the cohesiveness of the community and the total influence of the members of the community to estimate the overall score of a community (Section 5).

STEP 3 : Given the augmented query and the influential score function, our focus is now to retrieve top-r most influential communities relevant to the query. Since we are the first to propose the keyword-aware influential community search problem, and existing precomputation based approaches are not suitable to retrieve communities for any given query, we first present a basic solution named BASIC-EXPLORE followed by two efficient algorithms: PRUNED-EXPLORE and TREE-EXPLORE.

4. KEYWORD AUGMENTATION

In a social network, one's expertise in various fields can be represented by a collection of terms (words or phrases). There are millions of such terms in a large network and it is difficult for users to come out with the exact terms while raising a query. We propose a semantic keyword similarity model that can augment any term with relevant keywords. This model is used to extend the keywords in the attributed graph, and associate appropriate keywords for each term in the query.

4.1 Semantic keyword similarity model

Finding semantically related keywords of a term is not trivial. Basic preprocessing like removing whitespaces and stopwords are not sufficient. Two or multiple terms with a slight syntactic difference can indicate the same keyword (e.g., "error detection and error correction" and "error detection and correction"). Even two different terms can represent the same keyword (e.g., "AI" and "artificial intelligence") or can be semantically similar (e.g., "neural network" and "deep learning").

We adopt Word2Vec [30] model to generate an embedding vector of any given word. We train this model with a domain corpus (e.g., scientific publications in an academic domain) after stopword removal, tokenization, and lemmatization [34]. A keyword/term can be thought of as a phrase containing one or many words. In our approach, the representative vector is formed using the average of the embedding vectors of the constituent words. Now, given any two terms t_1 and t_2, we denote their embedding vectors as x_{t_1} and x_{t_2}, respectively. To estimate a similarity between these two terms, denoted as $S(t_1, t_2)$, we can use widely used cosine similarity of their embedding vectors x_{t_1}, x_{t_2} [32]. We also propose a new similarity, which yields an attributed graph of better quality.

indirect cosine: Given a term t, its vector V^t is denoted as $V^t = [(w_1^t, s_1^t), (w_2^t, s_2^t), \cdots, (w_L^t, s_L^t)]$, where w_i^t is the ith most similar word to x_t, s_i^t is the corresponding similarity score, and L is the number of similar terms of t.

Given two terms t_1 and t_2, we construct a vocabulary, U combining words of V^{t_1}, and V^{t_2}. Formally, $U = \{w : (w, s) \in V^{t_1} \cup (w, s) \in V^{t_2}\}$. To simplify our notation, let $U = \{w_1, w_2, \cdots, w_n\}$, where $n = |U|$. Now, we define another vector $SV^t = [s_1, s_2, \cdots, s_n]$, where s_i is the similarity score of term t to word $w_i \in U$, which can be found from V^t. If $(w_i, s_i) \notin V^t$, s_i is set to 0. Finally, $S(t_1, t_2)$ is calculated as the cosine similarity of SV^{t_1} and SV^{t_2}.

Finally, for any term t, we calculate its similarity with all the keywords in the given attributed graph, and find M-top

most relevant keywords X_t ranked based on the similarity scores. M is a system configurable parameter. By default, M is set to 10.

4.2 Constructing attributed graph

In attributed graph G^+, a vertex v is associated with a set of keywords. For all vertices, we extend each keyword t with its M-top most relevant keywords X_t using the semantic similarity model. For any keyword $w \in X_t$, the influence score of vertex v, $s_v(w) = s_v(t)$ since w and t are semantically similar.

4.3 Augmenting keywords for KICQ

A query $q(T, P)$ consists of a set of terms $T = \{t_1, t_2, \cdots, t_n\}$ and a predicate P. First, the semantic similarity model is used to augment each term t_i with the set of relevant keywords X_{t_i}. Then the system parameters r and k_{min} are used to formulate the keyword aware influential community query, $KICQ(X, P, r, k_{min})$ where $X = \{X_{t_1}, X_{t_2}, \cdots, X_{t_n}\}$.

5. INFLUENTIAL COMMUNITY MEASURES

We design a scoring function that considers connectivity, cohesiveness, influence of individuals and the community size, and assigns a score for ranking the candidate communities given a query.

First, for a given query, we redefine the influence of a vertex based on its relevance to the query. The query relevance score γ_v of a vertex v is estimated as follows: each vertex v in the attributed graph is annotated with keywords and their influence score for the corresponding keywords, i.e., $(w_i, s_v(w_i))$. To estimate the relevance score, $\gamma_v \in [0, 1]$, we need to consider the list of semantic keywords X and the predicate P in the $KICQ$ query. Formally, we use the following definition for computing γ_v:

$$\gamma_v = \mathbf{f}_{X_{t_i} \in X}[\mathbf{g}_{w \in X_{t_i}} s_v(w)] \tag{1}$$

Here, $\mathbf{f}$ and $\mathbf{g}$ are two aggregate functions: $\mathbf{g}$ combines the relevance of the vertex for the semantic keywords of a query term, and $\mathbf{f}$ combines the relevance scores in all terms considering the predicate P. We use $\mathbf{g}$ as the aggregate function, MAX; whereas we use $\mathbf{f}$ as MIN for AND predicate and MAX for OR predicate, respectively. MIN aggregate ensures that a vertex has high relevance to all the terms, while MAX only requires high relevance to any of the terms.

Now, we use a linear weighted summation of the cohesiveness and influences to calculate the overall score of a community. Let $H = (V_H, E_H)$ is a subgraph of attributed graph $G^+(V, E, A)$. If H is a community (connected component of maximal k-core), then the score of H is:

$$\zeta(H) = \beta \times \underbrace{\frac{k}{max\text{-}deg(G^+)}}_{\text{Cohesiveness score}} + (1-\beta) \times \underbrace{\frac{\sum_{v \in V_H} \gamma_v}{|V|}}_{\text{Influence score}} \tag{2}$$

Here, $max\text{-}deg(G^+)$ is the maximum degree of all vertices in G^+. Both the cohesiveness and the influence score of a community are normalized within $[0, 1]$, and the preference parameter $\beta \in [0, 1]$ defines the importance of one score relative to the other.

Since we model a community using connected k-core, connectivity and cohesiveness is ensured. $max\text{-}deg(G^+)$ and $|V|$ is constant for attributed graph G^+. Thus the influence score of community H depends on $\sum_{v \in V_H} \gamma_v$ which prefers large community with highly influential individuals. Also

as long as some low influential members do not disrupt the cohesiveness of the community, the score of this community is not penalized, which is the case in [27]. We acknowledge that such a measure is not unique, and other measures can be explored in the future. However, experiments using real datasets and the case study presented in Section 7.5 demonstrate that our proposed influence measure can capture cohesive communities with highly influential members.

6. ALGORITHMS FOR INFLUENTIAL COMMUNITY SEARCH

In this section, we present algorithms for finding r most influential communities from the attributed graph G^+ for a given $KICQ(X, P, r, k_{min})$. Since the notion of *influential community* changes with different sets of query keywords, existing pre-computation based approach [27, 25] cannot be adapted for this purpose.

6.1 A straightforward approach, BASIC-EXPLORE

A straightforward approach to answer $KICQ$ on a large graph is as follows. First, we extract the subgraph, which we call the *query essential subgraph*, G_q, containing vertices and edges that are relevant to the query. Then we find all the connected components of maximal k-core subgraphs for all possible values of k. Finally, we return the top r communities having the highest influential community scores as per Equation 2.

Finding G_q. The query essential subgraph, $G_q(V_q, E_q, \gamma)$ is a subgraph of the attributed graph $G^+(V, E, A)$ induced by V_q, the set of vertices with a non-zero query relevance score. In G_q, each vertex v is annotated with its relevance score γ_v, and E_q is the set of edges between any two vertices in V_q. To efficiently generate the G_q, we maintain an inverted index, where for each keyword w, a list IL_w of the vertices that contain w is stored. Thus, for a given $KICQ$ query, V_q can be obtained by,

$$V_q = \begin{cases} \bigcap_{X_{t_i} \in X}[\bigcup_{w \in X_{t_i}} IL_w], & \text{if } P = AND \\ \bigcup_{X_{t_i} \in X}[\bigcup_{w \in X_{t_i}} IL_w], & \text{otherwise} \end{cases} \tag{3}$$

After retrieving V_q, we compute the query relevance score of each vertex $v \in V_q$ (Equation 1) and retrieve E_q that denotes the connections between all pairs of vertices in V_q.

Finding k-cores and most influential communities. Algorithm 1 outlines the procedure BASIC-EXPLORE for finding r most influential communities from G_q. First, we compute core decomposition for all vertices in G_q using the $O(|E_q|)$ algorithm proposed by Batagelj et al. [2]. A priority queue Q is used to hold our solution. We initialize Q with r empty communities having 0 score. In our case, a community must be at least k_{min}-core. Again, the maximum cohesion factor of a community in G_q can be $max\text{-}deg(G_q)$, since there is no vertex in G_q with a higher degree. Thus we need to first find all connected components of maximal k-cores from G_q, where the value of k is in range $[k_{min}, max\text{-}deg(G_q)]$. Then, we compute the influential scores of each computed community, and finally, maintain the top-r communities in Q ordered by the scores of the communities.

Time complexity. Finding the relevant vertices and calculating their relevance score can be done in $O(|V_q| \times N_w)$

Algorithm 1 BASIC-EXPLORE (G_q)

1: compute core decomposition for all vertices in G_q
2: initialize a priority queue Q with r empty communities (score 0)
3: **for** $k = k_{min}$ to $max\text{-}deg(G_q)$ **do**
4:     H^k = maximal k-core in G_q
5:     CC^k = set of connected components in H^k
6:     **for** all $h(V_h, E_h) \in CC^k$ **do**
7:         $\zeta(h)$ = score of h
8:         **if** $\zeta(h) > r^{th}$ best score **then**
9:             Q.pop()
10:            Q.push(h)

---

time, where $N_w = \sum_{X_i \in X} |X_i|$ is the total number of relevant keywords. If the graph is implemented with adjacency list, E_q can be obtained in $O(|V_q|)$ time by taking union of adjacency list and V_q for each vertex. So, time complexity for computing G_q is $O(|V_q| \times N_w)$. Core decomposition of V_q is done in $O(|E_q|)$ time. The operations (push, pop) performed in the priority queue takes $O(\log r)$ time. So, the time required for initializing Q is $O(r \log r)$. Exploring a maximal k-core requires computing its connected components $(O(|V_q| + |E_q|))$, obtaining k-core vertices $(O(|V_q|))$, computing scores of each connected components, and updating the priority queue Q. For any community $h(V_h, E_h)$, the run-time for computing its score is bounded by $O(|V_h|) = O(|V_q|)$ (simplified). So, if N_k is the number of influential communities with cohesion factor k, then the runtime of exploring all k-cores is bounded by $O(max\text{-}deg(G_q) \times ((|V_q| + |E_q|) + N_k \times (|V_q| + log(r))))$. Considering $|V_q| > log(r)$, the bound can be simplified as $O(max\text{-}deg(G_q) \times |V_q|^2)$ for a dense graph [1]. Since, this dominates the time complexity of finding G_q, we can conclude that, the overall complexity of BASIC-EXPLORE is $O(max\text{-}deg(G_q) \times |V_q|^2)$.

6.2 Pruned exploration approach, PRUNED-EXPLORE

The major bottleneck of BASIC-EXPLORE is that it needs to explore all maximal k-cores, for different values of k, and find the connected components of each maximal k-core subgraph. Such exploration is computationally expensive for a large graph. Instead of directly exploring the subgraphs to compute the maximal k-core and its connected components (communities), we first estimate the upper bound score of the communities of the corresponding subgraph. This bound can be used to prune a large number of redundant subgraphs that cannot be a part of the top-r influential communities.

First, we find the query essential subgraph G_q, compute core decomposition, and initialize priority queue Q as described in Section 6.1. Now, we need to explore G_q to retrieve communities for all possible values of k. As discussed before, the value of k must be between k_{min} and $max\text{-}deg(G_q)$. We propose the following lemmas, which pave the foundation of our pruning.

Lemma 1. Let, $H(V_H, E_H)$ be a subgraph of G_q. For any community in H, the maximum influence score can be the sum of the query relevance scores of all vertices in H. Thus, without computing the vertices of k-core subgraph, we can calculate the upper bound of the score of any community in H for a particular value of k as follows.

$$\zeta_k^*(H) = \beta \times \frac{k}{max\text{-}deg(G^+)} + (1 - \beta) \times \frac{\sum_{v \in V_H} \gamma_v}{|V|} \quad (4)$$

Algorithm 2 PRUNED-EXPLORE (H, k)

Input: Subgraph of G_q $H = (V_H, E_H)$, cohesion factor k.
1: $min\text{-}deg(H) = \min\limits_{v \in V_H} (deg_H(v))$
2: **if** $min\text{-}deg(H) > k$ **then**
3:     $k = min\text{-}deg(H)$
4: H^k = maximal k-core in H
5: CC^k = set of connected components of H^k
6: **for** all $h \in CC^k$ **do**
7:     **if** actual score, $\zeta(h) > r^{th}$ best score **then**
8:         Q.pop()
9:         Q.push(h)
10:    **for** $k' = k + 1$ to $max\text{-}deg(G_q)$ **do**
11:        **if** upper bound score, $\zeta_{k'}^*(h) > r^{th}$ best score **then**
12:            PRUNED-EXPLORE(h, k')
13:            break

Lemma 2. If $H = (V_H, E_H)$ is a subgraph and $min\text{-}deg(H) = \min\limits_{v \in V_H} (deg_H(v)) > k$, then any community in H must be at least $min\text{-}deg(H)$-core.

According to Lemma 1, we can prune a subgraph if its upper bound score is lower than the rth best score of already retrieved communities from G_q. Moreover, Lemma 2 helps us to avoid the computation of certain cores from G_q.

Now, we develop a recursive procedure PRUNED-EXPLORE to search for influential communities in G_q. Algorithm 2 outlines the procedure. Initially, PRUNED-EXPLORE(G_q, k_{min}) is called to extract communities with minimum cohesion factor. In later steps, the procedure is recursively called to extract communities with higher cohesion factors.

Let us consider that we want to find communities with cohesion factor k, from a subgraph $H(V_H, E_H)$ of G_q. In lines 1-3, we determine the minimum degree of the vertices in H, $min\text{-}deg(H)$. If $min\text{-}deg(H) > k$, we set $k = min\text{-}deg(H)$ and directly compute such k-cores (according to Lemma 2). In lines 4-5, we find the set of connected components of maximal k core of H, denoted by CC^k. The loop in line 6 runs for each connected component. We update the priority queue if any connected component's score is higher than the current top-r communities in lines 7-9. We explore the connected component for higher values of k in lines 10-13. We use Lemma 1 to prune exploration for the values of k for which the upper bound of the score is lower than the rth best community. When the procedure terminates, the queue holds the final top-r communities.

6.3 Keyword indexed tree exploration, TREE-EXPLORE

Though the above PRUNED-EXPLORE can prune a large number of subgraphs based on the derived upper bounds, it still explores subgraphs and their connected components with low cohesiveness, which usually do not contain the most influential communities. This exploration can be costly, especially in a scenario where the query essential graph, G_q, turns out to be very large. So, we propose a novel index, namely *keyword indexed core-label tree* (KIC-tree), that precomputes and organizes the connected components of maximal k-core subgraphs hierarchically with computed upper bound of influence scores for each keyword.

The key idea of our KIC-tree based $KICQ$ comes from the following observations:

(i) Top communities are structurally cohesive and thereby can be retrieved by exploring the subgraphs of higher cohesion factors. Thus, if k-cores are precomputed, disregarding the associated keywords, we can still prune the subgraphs with low k value.

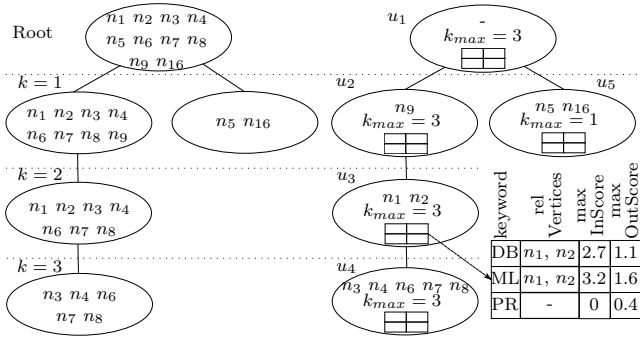[1] $N_k < |V_q|$ and for any dense graph $G(V, E)$, $|E|$ is $O(|V|^2)$

Figure 3: KIC-tree for the subgraph (shaded) in Figure 1.

(ii) Communities are represented using connected maximal k-cores which are nested, i.e., by definition, a $(k + 1)$-core is also a k-core ($k \geq 0$). This property helps to store all the connected components of maximal k-cores in compressed tree-based structures as shown in previous works `ICP-index` [27], `CL-tree index` [13].

(iii) We can compute the upper bounds for both the components: influence and cohesiveness, of the the scoring function, and use these upper bounds to prune the search space during query time.

We first discuss the basic structure of the `KIC-tree` index. Then we present the upper bounds for individual keywords and aggregate them for a set of keywords and predicates (in $KICQ$) for an upper bound score of a node. We also show how the cohesiveness score can be bounded based on a pre-computed structure alone. Finally, we present our `TREE-EXPLORE` algorithm for influential community search using the `KIC-tree`. In this section, we use the term "node" to exclusively indicate a tree node.

6.3.1 KIC-tree index

The `KIC-tree` index organizes the connected components of k-cores into a space-efficient tree structure. We adopt the concept of compressed tree based structure of previous works (e.g., `CL-tree index` [13]), and augment the structure with derived bounds to prune the search space.

Figure 3 shows an example `KIC-tree` for the subgraph shown as the shaded region in Figure 1. The left shows the hierarchical representation of all maximal k-core connected components in the subgraph. We refer this tree as the *uncompressed tree*. The right figure shows `KIC-tree` index, a more compact representation of the left tree, which removes the graph vertices present in its descendant nodes ensuring that each graph vertex appears exactly once.

Let u be a `KIC-tree` node and $subtree(u)$ be the subtree rooted at u. The structure of u is as follows:

(i) k, the cohesion factor; (ii) $vertexSet$, the set of compressed graph vertices at node u; (iii) $childNodes$, the set of child nodes of u; (iv) k_{max}, the maximum cohesion factor of any connected component contained by the $subtree(u)$; (v) $iList$, an inverted list containing the upper bounds of influence scores for all keywords appeared in $subtree(u)$.

For each keyword w that appears in $subtree(u)$, the inverted list $u.iList[w]$ contain the following elements:

(i) $relV$, a set of graph vertices in $u.vertexSet$ containing the keyword w;

(ii) $maxKNScore$, the upper bound of influence score component by only considering keyword w in a community (i.e., a connected component) contained by the $subtree(u)$,

where the community must include at least one vertex present in node u containing the keyword w;

(iii) $maxKDScore$, the upper bound of influence score component by only considering keyword w of a community contained by the $subtree(u)$, where the community does not include any vertex from $u.vertexSet$ (i.e., all vertices of the community come from the descendent nodes of u).

We compute $maxKNScore$ and $maxKDScore$ as follows.

For a node u, let $childV$ be the set of graph vertices stored at the descendent nodes of u, and $allV$ be the set of graph vertices at $subtree(u)$ (i.e., all the vertices in node u and its descendent nodes). If u is a leaf node, $u.childV = \emptyset$. Otherwise, $u.childV = \bigcup_{p \in u.childNodes} p.vertexSet$. On the other hand, in all cases, $u.allV = u.vertexSet \bigcup u.childV$.

Now, if there is no relevant graph vertex in node u for keyword w, then we set $maxKNScore$ as 0. Otherwise, the upper bound is the sum of influence scores of all graph vertices in $u.allV$. Formally,

$$u.iList[w].maxKNScore = \begin{cases} 0, \text{if } u.iList[w].relV = \emptyset \\ \sum_{v \in u.allV}(s_v(w)), \text{otherwise} \end{cases}$$

Here, $s_v(w)$ is the influence score of vertex v for keyword w.

Now, $maxKDScore$ is the maximum influence score component among the communities represented by the descendant nodes of u. $u.iList[w].maxKDScore = 0$ if u is a leaf node. Otherwise, we can use the computed values of $maxKNScore$ to compute the $maxKDScore$ as follows.

$$u.iList[w].maxKDScore = \max_{p \in u.childNodes} p.iList[w].maxKNScore$$

Figure 3 (right) shows an example tree, where the table inside the ellipse represents the $iList$ of the corresponding node. For simplicity, we only show the $iList$ for node u_3.

6.3.2 Complexity analysis for index construction:

We use the `advanced` method proposed by Fang et. al. [13] that compresses the tree and for each node u, computes $u.iList[w].relV$ for all the relevant keywords of u. The time complexity of this method is $O(|E| \times \alpha(|V|))$, where $\alpha(|V|)$, the `inverse Ackermann function`, is less than 5 for all remotely practical values of $|V|$. For each $iList[w]$ entry, we also need to compute the two upper bounds $maxKNScore$ and $maxKDScore$. If A_{max} is the maximum number of keywords associated with a graph vertex, the time complexity for computing $maxKNScore$ is $O(A_{max} \times |V|)$. Computing $maxKDScore$ for a node u only requires visiting its $childNodes$ which is non-dominant. So, overall time complexity for index construction is $O(|E| \times \alpha(|V|) + A_{max} \times |V|)$.

In $iList$, we need additional space to store two upper bound scores (constant space) for each keyword. The space cost is still dominated by storing $relVertices$ in $iList$. So, the space complexity remains $O(\bar{A} \times |V|)$ as in [13], which is proportional to the graph size.

6.3.3 Computing upper bound scores for a query

Given a $KICQ(X, P, r, k_{min})$ query, we need to compute an upper bound influence score of a community denoted by S_{inf} and the maximum possible cohesiveness score of that community, S_k by using the precomputed upper bounds in `KIC-tree`. Then the upper bound of the total score of that community can be computed as $maxScore = \beta \times S_k + (1 - \beta) \times S_{inf}$ (as in Equation 2).

We define two upper bounds for the communities inside $subtree(u)$: (i) $maxNodeScore$, the maximum possible score

of any community that can be exclusively found by exploring the connected k-core stored at node u and (ii) $maxDesScore$, the maximum possible score of any community that can be found by exploring the descendant nodes of u.

Computing $maxNodeScore$**:** For any community contained exclusively by node u, there must be at least one vertex v that is stored at u. Now, for any vertex v exclusive to node u, $u.k$ is the maximum core number. So, a subgraph containing v can be at most $u.k$-core (irrespective of any keyword) and the upper bound of cohesiveness score of any community contained by the node can be computed as $S_k = u.k/max\text{-}deg(G^+)$.

Now, for each keyword w, $u.iList[w].maxKNScore$ already defines the upper bound of influence score component for any community in the subgraph exclusively contained by node u (considering the single keyword w). We combine these bounds for considering all the keywords in the $KICQ$ query and compute the maximum influence score as:

$$S_{inf} = \frac{1}{|V|} \times \mathbf{F}_{X_{t_i} \in X}(\textstyle\sum_{w \in X_{t_i}} u.iList[w].maxKNScore)$$

Here $\mathbf{F}$ is an aggregate function that combines the influence scores of the community for multiple terms depending on the predicate P and division by $|V|$ normalizes the score within [0,1]. For the queries with OR predicate, a top community can be formed by joining multiple communities pre-computed for a single term, and these communities may have disjoint vertex set. So, it is safe to consider $\mathbf{F}$ as a **summation** aggregate. For the same reason, $\sum$ is explicitly used to combine the semantic keywords of a term. Again, for the queries with AND predicate, any graph vertex forming a community for a single term must be present in communities of other terms as well. So, $\mathbf{F}$ can be safely considered as **minimum** aggregate.

Computing $maxDesScore$**:** For any community contained by the descendant nodes of u, the maximum cohesion factor is $u.k_{max}$ and the upper bound of cohesiveness score is $S_k = u.k_{max}/max\text{-}deg(G^+)$.

Again, for keyword w, $u.iList[w].maxKDScore$ already defines the upper bound of influence score component for any community contained by the descendant nodes. We combine these bounds for considering all the keywords in the $KICQ$ query and compute the maximum influence score similarly as computing $maxNodeScore$, i.e.,

$$S_{inf} = \frac{1}{|V|} \times \mathbf{F}_{X_{t_i} \in X}(\textstyle\sum_{w \in X_{t_i}} u.iList[w].maxKDScore)$$

6.3.4 TREE-EXPLORE *algorithm*

We follow a *best-first* exploration strategy. Since the leaf nodes contain the communities with high cohesiveness while nodes near root contain communities with low cohesiveness, we explore the KIC-tree in a post-order manner. Likewise the previous exploration algorithms (e.g., PRUNED-EXPLORE), a priority queue Q initialized with r empty communities is used to store the results. The exploration algorithm, which we call TREE-EXPLORE is developed based on the following pruning techniques:

(i) Subtree pruning: For any node u, we examine the $u.maxDesScore$ before visiting its children. If it is less than the rth best score, then none of the communities to be found in the descendent nodes can score higher than the current r^{th} top community. Therefore, we can skip visiting the descendant nodes of u.

(ii) Node pruning: Before exploring the pre-computed connected k-core subgraph at any node u, we examine the $u.maxNodeScore$. If it is less than the rth best score, we

---

Algorithm 3 TREE-EXPLORE (u, U)

Input: Tree node u, query relevant nodes U.
1: **if** u is an internal node **then**
2:     Compute influence score component S_{inf} and cohesiveness score component S_k for $u.maxDesScore$
3:         $u.maxDesScore = \beta \times S_k + (1 - \beta) \times S_{inf}$
4:     **if** $S_{inf} > 0$ and $u.maxDesScore > r^{th}$ best score **then**
5:         **for** each $p \in (u.childNodes \cap U)$ **do**
6:             TREE-EXPLORE(p, U)
7: **if** $u.k < k_{min}$ **then**
8:     return
9: Compute influence score component S_{inf} and cohesiveness score component S_k for $u.maxNodeScore$
10: $u.maxNodeScore = \beta \times S_k + (1 - \beta) \times S_{inf}$
11: **if** $S_{inf} = 0$ or $u.maxNodeScore < r^{th}$ best score **then**
12:     return
13: $u.V_{rel}$ = compute relevant graph vertices in the subtree rooted at u
14: Compute query relevance score of all vertices in $u.V_{rel}$
15: Compute $u.E_{rel}$, the edges among $u.V_{rel}$
16: Construct subgraph $H(u.V_{rel}, u.E_{rel})$, each vertex annotated with relevance score
17: MODIFIED-PRUNED-EXPLORE$(H, k_{min}, u.k)$

---

can safely prune this exploration.

Algorithm 3 outlines the pseudocode for the KIC-tree traversal. The inverted list that we have used to find the G_q is adopted for computing U, the set of tree nodes relevant to a query. Initially, the recursive procedure TREE-EXPLORE(u, U) is called with u being the root of the KIC-tree.

For any internal node u, we first compute and examine the influence score component, S_{inf}, and the cohesiveness score component, S_k of $u.maxDesScore$. If S_{inf} is 0, the descendants of u do not contain any graph vertex relevant to the query, and therefore we do not need to visit subsequent nodes across the subtree. If $S_{inf} > 0$ and $u.maxDesScore$ is greater than the current r^{th} best score, then we visit its children (lines 1-6).

Now we want to explore the pre-computed connected $k - core$ subgraph represented by node u. If the cohesion factor k in node u is less than k_{min}, then we prune exploring the subgraph. Otherwise, we compute the influence score component (S_{inf}) and the cohesiveness score component (S_k) of $u.maxNodeScore$. If S_{inf} is 0, then the node does not contain any graph vertex relevant to the query, and we can safely skip exploring the subgraph. Again, we skip the exploration if $u.maxNodeScore$ is less than the current r^{th} best score (lines 7-12).

If the exploration of the connected k-core subgraph cannot be pruned, we first need to find all the relevant graph vertices, $u.V_{rel}$ present in the subgraph (line 13). Since KIC-tree compresses these graph vertices by removing ones present at descendant nodes, we need to decompress in a bottom-up manner. At any node u, the relevant graph vertices $u.V_{rel}$ can be computed like the vertices in QEG (Equation 3) just by replacing IL_w with $u.iList[w].relV$. For any internal node u, we need to add the relevant graph vertices in child nodes to $u.V_{rel}$.

Now we compute the relevance score of each vertex $v \in u.V_{rel}$ (Equation 1) and then compute the edges among these vertices, thereby construct the subgraph $H(u.V_{rel}, u.E_{rel})$ (lines 14-16).

The procedure MODIFIED-PRUNED-EXPLORE(H, k, k_{max}) is a slightly modified version of the procedure PRUNED-EXPLORE(H, k) that takes an extra argument k_{max}, the maximum value of cohesion factor for sub-graph H. Lines 10-13 in Algorithm 2 are replaced by the following:

10: **for** $k' = k + 1$ to k_{max} **do**

11:      **if** $\zeta_{k'}^*(h) > r^{th}$ best score **then**
12:          MODIFIED-PRUNED-EXPLORE(h, k', k_{max})
13:          break

Since no graph vertex at u belongs to any k-core with cohesion factor higher than $u.k$, here $k_{max} = u.k$. Initially, $k = k_{min}$. So, MODIFIED-PRUNED-EXPLORE$(H, k_{min}, u.k)$ is called to explore the subgraph H (line 17).

7. EXPERIMENTAL STUDY

In this section we present experiments to evaluate the performance of our proposed algorithms.

7.1 Experimental setup

All the community search algorithms are implemented in JAVA. Experiments were run on a virtual environment of OzSTAR[2] supercomputer with two cores of Intel Gold 6140 CPU @ 2.30 GHz 2.30GHz, 192 GB RAM, and 400 GB SSD. We assume that the graph and all the indexes will fit in the memory. For the simplicity of presentation, we use shorter names for our algorithms: BASIC, PRUNE, and TREE to represent BASIC-EXPLORE, PRUNED-EXPLORE, and TREE-EXPLORE respectively. We present the average results of 100 queries.

We use two large real datasets: OAG (Open Academic Graph)[3] [39] and Gowalla[4] that reflect the real life application scenarios. In OAG dataset, we represent first 1 million authors as vertices and $15,677,940$ co-authorship relations among the authors as edges. We choose $1,000$ most frequent author-provided keywords as the set of keywords for the attributed graph and then apply our semantic similarity model to extend the keywords as mentioned in Section 4.1. In Gowalla dataset, users and friendship among them are modeled as vertices and edges, and the location ids are considered as keywords. There are $407,533$ vertices, $2,209,169$ edges, and $2,727,464$ keywords in this attributed graph. For both datasets, the influence score of a user for a certain keyword is modeled as the user's percentile rank considering the number of citations or check-ins. To generate a query for OAG datasets, first, we choose 1-3 query terms from the most frequent $10,000$ author-provided keywords, and then augment each keyword with its semantically similar keywords using our semantic keyword similarity model. For Gowalla dataset, we choose a set of locations within a range of 5km as query terms.

We vary different parameters as shown in Table 2. When one parameter is varied, other parameters are fixed at their default values.

We have uploaded the constructed attributed graphs in a public repository[5]. The repository also contains a detailed description of datasets, query setting, and the semantic similarity model and its evaluation.

7.2 Evaluation of semantic similarity model

In Section 4.1, we presented two approaches for finding semantic similarity between two terms or keywords. Here, we empirically evaluate which approach is the most effective. As the ground truth, we use widely used semantic similarity measure [37] that is based on the intrinsic information content of two concepts in a given taxonomy. As the taxonomy, we use the taxonomy provided by the 2012 ACM

Parameter	Range	Default
Dataset	OAG, Gowalla	OAG
Dataset size (vertices)	$300K$, $500K$, $700K$, $900K$, $1M$	$500K$
Number of keywords	100, 250, 500, 750, 1000	1000
β	any real value within [0.0, 1.0]	0.60
r	any integer within [1, 5]	3
k_{min}	any integer within [2, 50]	10

Table 2: Parameters for experimental analysis

Computing Classification System[6] that contains 2,113 topics and organizes them hierarchically based on relevance (e.g., "clustering" is a sub-topic of "data mining"). In our context, each topic in the taxonomy can be seen as a keyword or a term, and each of our proposed similarity measures can be considered as a ranker that finds the most similar topics to any topic in the taxonomy. So, we adopt Normalized Discounted Cumulative Gain ($NDCG$) [42], which is a widely accepted performance measure of ranking systems.

First, given the taxonomy τ, we give the ground truth formula of the similarity between two topics t_1 and t_2 proposed in [37]: $sim_{jcn}(t_1, t_2) = 1 - [d_{jcn}^\tau(t_1, t_2)/2]$, where $d_{jcn}^\tau(t_1, t_2)$ denotes a distance metric between t_1 and t_2 as $d_{jcn}^\tau(t_1, t_2) = IC^\tau(t_1) + IC^\tau(t_2) - 2 \times IC^\tau(lcs(t_1, t_2))$ [19]. Here, $lcs(t_1, t_2)$ is the "least common subsumer" in τ that subsumes t_1 and t_2. Finally, $IC^\tau(t)$ indicates the information content of topic t in τ, calculated by $\log(\frac{|sc^\tau(t)|+1}{|\tau|})/\log(\frac{1}{|\tau|})$ as in [37]. Here, $sc^\tau(t)$ is the set of subsumed topics of t and $|\tau|$ is the total number of topics in τ.

Second, given a topic t in τ, our task is to rank the M-top similar topics $\{w_1, w_2, \cdots, w_M\}$ in τ. The ranking goodness is evaluated by $NDCG@M$ that gives more importance on the ranking of a more relevant entity than the ranking of entities with lower relevance.

To obtain a vector representation of each topic in τ, we use Google's pre-trained word2vec model[7] that includes embedding vectors for a vocabulary of 3 million words and is trained on Google News dataset covering academic research. The length of the embedding vector is 300.

Table 3 shows the comparison among the two similarity metrics in terms of $NDCG@50$, $NDCG@20$, $NDCG@10$. We choose indirect cosine that outperforms cosine metric.

Metric	cosine	indirect cosine
NDCG@50	0.528	**0.542**
NDCG@20	0.537	**0.607**
NDCG@10	0.573	**0.633**

Table 3: Performance results of similarity metrics

In the indirect cosine similarity measure (see Section 4.1) there is a challenge in determining a good value for L. To examine this, we use two measures. The first is *word coherence*, $WC^L(V^t)$, that indicates how coherent the L-top words in vector V^t are. The more coherent it is, the better we can represent the set of keywords. We define the word coherence as the average pairwise cosine similarity of the L-top words. Also, in a sense, each term is a cluster containing its relevant words. So we use *Davies-Bouldin Index* [11] as the second measure that optimizes two criteria: (1) minimizing intra-distance between words and the centroid, and (2) maximizing inter-distance between keywords.

Values closer to zero indicate a better clustering. A small value of L usually contains words with high coherence and better clustering, but it is tough to find similar keywords. A large value contains words with low coherence and worse clustering, but it is easier to find similar keywords. The desired value of L should be able to find a sufficient number of similar keywords, and provide high word coherence and low Davies-Bouldin index. By experimental evaluation[8], we found that $L = 15$ provides a nice trade-off being able to retrieve at least 10 similar keywords given a term in 98% cases.

7.3 Evaluation of KICQ

The `OAG` dataset is enriched with metadata from millions of articles, and better fits the application scenarios of our study, and thus we use it as the default dataset, unless otherwise stated, to demonstrate the performance of our proposed algorithms. First, we evaluate how different parameters (Table 2) affect the efficiency and effectiveness of the competitive algorithms. Then we compare the performance of our algorithm with the state-of-the-art influential community search algorithm [27].

7.3.1 Performance evaluation

In this section, we show the scalability, sensitivity of different parameters, memory requirement, and the cohesiveness of the retrieved communities by running a wide range of experiments.

KICQ processing time: In this set of experiments, we evaluate and compare the runtime of our proposed algorithms.

Varying the dataset size. To show the scalability, we consider different size of `OAG` dataset by varying the number of vertices and thereby edges. Figure 4 shows that with the increase in the number of vertices, runtime also increases. For AND predicate, the efficiency is nearly the same for all techniques because the query essential graph (in both `BASIC-EXPLORE` and `TREE-EXPLORE`) is small since most of the vertices are filtered out. The advantage of pruning few vertices is ruled out by the overhead of exploring the large tree in `TREE-EXPLORE`. However, for more time consuming OR predicate, `TREE-EXPLORE` is the most efficient approach, almost 1-5 times faster than the `BASIC-EXPLORE`. Also, `PRUNED-EXPLORE` and `TREE-EXPLORE` scales much better than `BASIC-EXPLORE`.

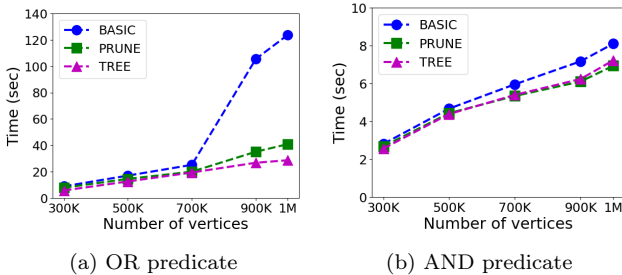(a) OR predicate   (b) AND predicate

Figure 4: Query processing time for varying dataset size

Varying k_{min}. Figure 5 shows how the query processing time is affected by parameter k_{min}. None of the `BASIC-EXPLORE` and `PRUNED-EXPLORE` algorithms are significantly affected by the value of k_{min}. However, if k_{min} is set to a high value,

`TREE-EXPLORE` does not need to explore the tree nodes representing k-cores with lower k values. This enables `TREE-EXPLORE` to skip a larger part of the tree since most of the vertices in the `OAG` dataset has degree less than 10 making `TREE-EXPLORE` significantly faster for higher k_{min} values.

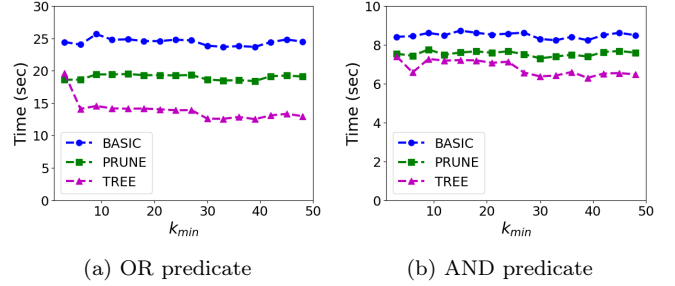(a) OR predicate   (b) AND predicate

Figure 5: Query processing time for varying k_{min}

Varying r. Figure 6 demonstrates how the query processing time is affected by the number of communities to be retrieved (query parameter r). Both `BASIC-EXPLORE` and `PRUNED-EXPLORE` compute core decomposition once on the entire query essential graph. However, `TREE-EXPLORE` performs the core decomposition on-demand basis. `BASIC-EXPLORE` is not affected by the value of r since it does not use any pruning based on the retrieved communities. `PRUNED-EXPLORE` prunes some expansion based on top-r score, but the query processing time is not noticeably affected. The effect is more substantial in `TREE-EXPLORE`. A significant part of the graph does not require core decomposition if r is small. If r is high, the algorithm can only prune a few tree nodes, but the advantage is ruled out by the overhead of decompressing the graph vertices inside a tree node. However, for small values of r, `TREE-EXPLORE` significantly outperforms the other algorithms, especially for OR predicate where query processing time is markedly higher than the AND predicate.
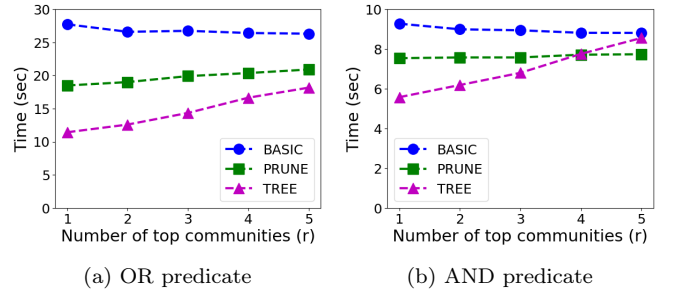

(a) OR predicate   (b) AND predicate

Figure 6: Query processing time for varying r

Index size: Figure 7 shows how the size of the graph and corresponding `KIC-tree` index increase with the increasing number of vertices and keywords. The result conforms to the complexity analysis demonstrated in Section 6.3.2. Index size is linear to both the number of vertices and number of keywords if one of these remains unchanged. Also, index size is bounded by the graph size.

Structural cohesiveness: We use popular structural cohesiveness metrics diameter, density, average degree, and clustering coefficient [15] to measure the quality of the communities retrieved by our approach. These measures mostly depend on the community models (e.g., k-core, k-truss) as discussed in a survey of community search [15]. They prefer the k-core model because of its high efficiency with minimal
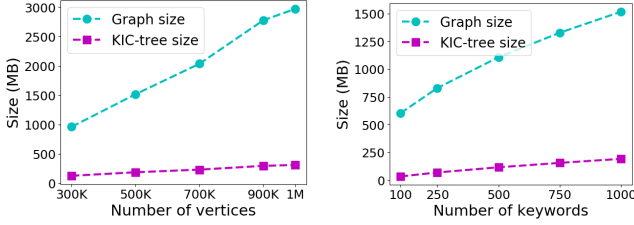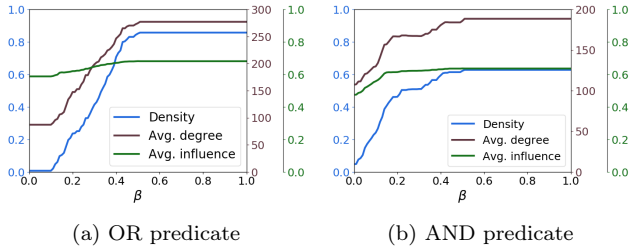
Figure 7: Index size for varying vertices and keywords

sacrifice in structural cohesiveness. By analyzing the cohesiveness measures in Table 4, we claim that our algorithms can retrieve cohesive communities with a small diameter.

Dataset	Density	Average Degree	Clustering Coefficient	Diameter
OAG	0.621	177.897	0.708	2.12
Gowalla	0.579	7.484	0.881	2.70

Table 4: Structural cohesiveness measures

Setting the value of β: We first run experiments to find an appropriate β, which balances the weight of connectivity and individual influence (query relevance score). For larger β, top communities are likely to show more cohesiveness and discard communities with influential individuals but less connectivity. When β is smaller, the top communities might incline to individual influence than cohesiveness. To find a good choice of β, we consider the network structure, that is: (1) we plot structural cohesiveness measures (i.e., density and average degree [14]), and (2) use the average influence score of the members for different β values as shown in Figure 8.

(a) OR predicate   (b) AND predicate

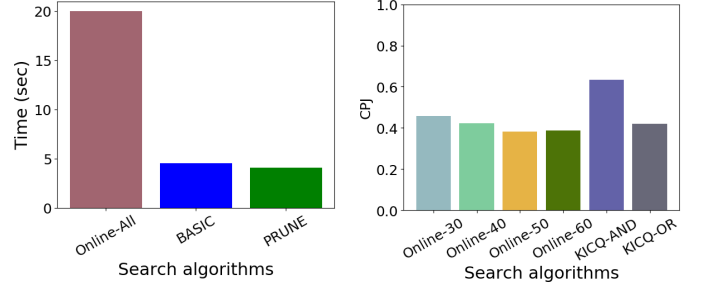Figure 8: Performance measures for varying β values

For OAG, communities with higher cohesiveness seem to contain influential individuals. This can be easily explained by the fact that an author who has co-authorship with a large number of authors is likely to have a strong influence in her field of studies. So, for OAG, we choose a high value of β (i.e., 0.6). Note that, considering the influence of communities is still essential since it is the tie-breaker between two communities with the same cohesiveness.

7.3.2 Comparison with state-of-the-art

We choose Online-All [27] as the state-of-the-art approach as they find influential community in non-attributed graphs.

Efficiency: To compare our algorithms with Online-All, we construct queries with a single keyword (as it does not support keywords) and compute the query essential graph, which is the input graph to Online-All. We also need to input the cohesiveness parameter k in Online-All. For this, we only consider the top community ($r = 1$), and the value of k in the top community returned by our approach is fed to Online-All. We do not consider the other algorithms in [27] since they use pre-computation which cannot be adopted

for our problem. For fair comparison, we do not consider TREE-EXPLORE algorithm since it uses pre-computed index. We show the query processing times of these algorithms in Figure 9a. Our approaches are *four times* faster compared to the Online-All algorithm.

(a) Query processing time   (b) Keyword cohesiveness

Figure 9: Effectiveness and efficiency comparison

Cohesiveness: To compare the keyword cohesiveness, we use community pair-wise Jaccard (CPJ) metric proposed in [13]. CPJ measures the similarity of the members of top communities in terms of keywords. We form the queries as described in Section 7.1 to evaluate our approach. OnlineAll cannot process queries with keywords; rather, it requires a cohesiveness parameter k as input. For the simplicity of presentation, we denote our approach as KICQ-AND, KICQ-OR for AND, OR predicates, respectively. Online-x denotes OnlineAll with parameter $k = x$. The comparison is presented in Figure 9b. For the communities returned by our approach with AND predicate, keyword cohesiveness is 1.5 times higher than OnlineAll, while for OR predicate CPJ is similar. This is expected as for OAG, two vertices are only connected when corresponding authors publish a paper together, and they also share common keywords. For this reason, OnlineAll finds communities with good CPJ value. However, this might not the case to other social networks (e.g., Gowalla, Twitter). Note that our approach and [27] use the same community model (i.e., k-core), and the structural cohesiveness is similar.

7.4 Experiments with Gowalla dataset

We conduct experiments on Gowalla dataset to show that our algorithms can handle large number (millions) of keywords. For these experiments, we carefully craft the queries as described in Section 7.1, and use $\beta = 0.5$, $r = 3$, $k_{min} = 5$ as default values. First, we report the cohesiveness measures in Table 4 and observe that our approach can retrieve cohesive communities with a small diameter for Gowalla dataset as well. We also compare the effectiveness and efficiency of the communities retrieved by our approach, and Li et al. [27]. The query setup and parameter settings are done in the same way as in the OAG dataset. The query processing time and keyword cohesiveness of these two approaches are presented in Figure 10. The number of relevant vertices is very small for queries in this dataset. The average query processing time is around 2 ms for all the algorithms. However, our key observation is that unlike OAG, in a dataset where connectivity is not related to keywords, the Online-All fails to address the keyword cohesiveness of the communities, while our approach returns communities considering both structural and keyword cohesiveness. The results show that our

approach returns communities with significantly higher (approx. 15 and 5 times for AND and OR predicate respectively) keyword cohesiveness than the `Online-All`.
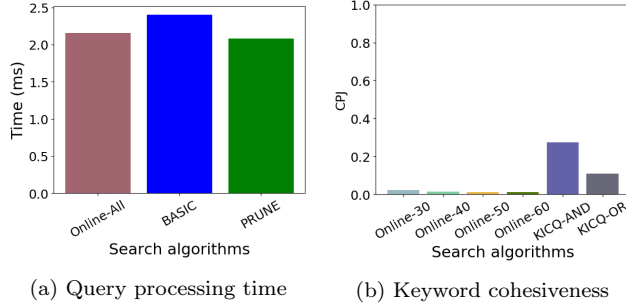

(a) Query processing time   (b) Keyword cohesiveness

Figure 10: Effectiveness and efficiency comparison

7.5 A case study

We use a small dataset of co-author network from Arnet-Miner[9] [39] to study the quality of retrieved communities. The dataset contains 5,411 vertices and 17,477 edges. Each vertex represents an author annotated with fields from eight different research areas: Data Mining (DM), Web Services (WS), Bayesian Networks (BN), Web Mining (WM), Semantic Web (SW), Machine Learning (ML), Database Systems (DS), and Information Retrieval (IR), where the influence score in each field depends on the number of publications in that field. There is an edge between two authors if they publish at least two papers together.

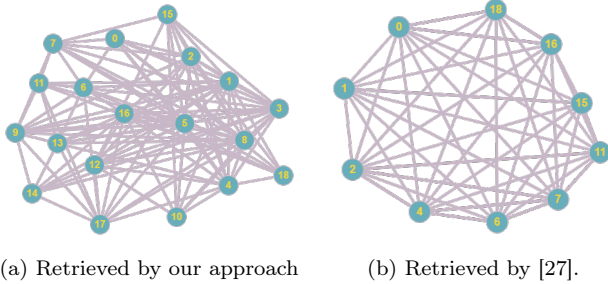(a) Retrieved by our approach   (b) Retrieved by [27].

Figure 11: Retrieved top communities for DS.

Note that [27] also conducted a case study on this dataset. Figure 11 shows the top community in DS retrieved by our approach (Figure 11a) and by [27] (Figure 11b). The top community returned by our approach is 8-core and thus we compare the result of [27] for $k = 8$. The details, i.e., h-index and number of citations of each author in our community are shown in Table 5. Among them, the authors who are not included in [27]'s community are shown in bold text. Due to the minimum score modelling, they missed out some of the top authors in this area including Rakesh Agrawal who was awarded the most influential scholar in the research area of Database Systems (DS) in Aminer [10]. Our approach keeps him in the community as we did not exclude a relatively less influential (with good connectivity) author Laura M. Haas. When Laura is not included in the community, Rakesh Agarwal is connected to less than 8 authors in the community, which turns out to be a non 8-core community. Since in the minimum weight modelling of [27], inclusion of a low influential member like Laura M. Haas significantly reduces the

score of the entire community, the resultant community no longer remains the top community for $k = 8$. These findings show the effectiveness of our problem formulation and score function modelling.
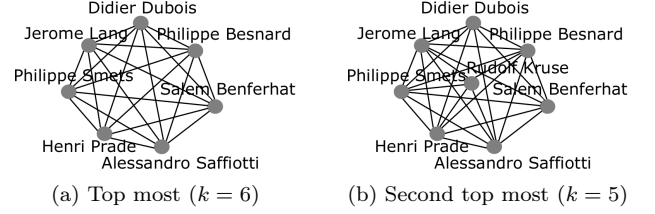

(a) Top most ($k = 6$)   (b) Second top most ($k = 5$)

Figure 12: Top communities for BN OR DM.

Figure 12 presents two top communities for "BN OR DM" returned by our algorithms. The top most community is fully connected and contains highly influential authors like Didier Dubois (h-index: 125, citations: 82,295), Henri Prade (h-index: 119, citations: 78,700). The second top most community also contains all the authors from top-1 community, but the inclusion of another author Rudolf Kruse (h-index: 54, citations: 16,829) increases the contribution of individual scores, but decreases the cohesiveness of the community resulting in a lower total score than the first one. This shows the flexibility and the trade-off capability among parameters while searching for the communities.

Vertex Id	Author Name	h-index	Citations
0	Hector Garcia-Molina	138	90,220
1	David Maier	65	36,687
2	David J. DeWitt	89	38,770
3	**Philip A. Bernstein**	**80**	**37,823**
4	Michael Stonebraker	72	26,153
5	**Michael J. Franklin**	**34**	**7,746**
6	Serge Abiteboul	80	35,950
7	Jennifer Widom	101	63, 641
8	**Joseph M. Hellerstein**	**90**	**43,207**
9	**Alon Y. Halevy**	**103**	**47,228**
10	**Jim Gray**	**81**	**46,884**
11	Gerhard Weikum	88	34,028
12	**Jeffrey F. Naughton**	**76**	**22,963**
13	**Yannis E. Ioannidis**	**59**	**15,017**
14	**Laura M. Haas**	**49**	**12,834**
15	Stefano Ceri	77	29,506
16	Michael J. Carey	59	16,451
17	**Rakesh Agrawal**	**108**	**124,595**
18	Umeshwar Dayal	62	26,527

Table 5: Top communities by our approach in DS.

8. CONCLUSION

In this paper, we have introduced the keyword-aware influential community query ($KICQ$) that finds top-r most influential communities from an attributed graph, which has many practical applications. First, we have designed the $KICQ$ as a set of query terms conjoining with predicates (AND or OR) that enables a user to search for influential communities from an attributed graph enriched by our proposed word-embedding based keyword similarity model. We have also proposed an influence measure for a community that considers both the cohesiveness and influence of individuals in the community. To answer the $KICQ$ efficiently, we have developed two algorithms based on the derived upper bounds and results from already explored subgraphs. Our experimental results and case study show that our approach outperforms the state-of-the-art approach in both efficiency (up to 4 times faster) and effectiveness (up to 15 times higher keyword cohesiveness).

References

N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data mining and knowledge discovery*, 29(5):1406–1433, 2015.

V. Batagelj and M. Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

F. Bi, L. Chang, X. Lin, and W. Zhang. An optimal and progressive approach to online search of top-k influential communities. *PVLDB*, 11(9):1056–1068, 2018.

L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou. Contextual community search over large social networks. In *ICDE*, pages 88–99. IEEE, 2019.

S. Chen, R. Wei, D. Popova, and A. Thomo. Efficient computation of importance based communities in web-scale networks using a single machine. In *CIKM*, pages 1553–1562. ACM, 2016.

Y. Chen, Y. Fang, R. Cheng, Y. Li, X. Chen, and J. Zhang. Exploring communities in large profiled graphs. *IEEE TKDE*, 2018.

J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *TODS*, 36(4):21, 2011.

S. Chobe and J. Zhan. Advancing community detection using keyword attribute search. *Journal of Big Data*, 6(1):83, 2019.

W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *The VLDB Journal*, Jul 2019.

S. Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

X. Huang and L. V. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.

J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

Y. Jiang, C. Jia, and J. Yu. An efficient community detection method based on rank centrality. *Physica A: statistical mechanics and its applications*, 392(9):2182–2194, 2013.

A. Khan, L. Golab, M. Kargar, J. Szlichta, and M. Zihayat. Compact group discovery in attributed graphs and social networks. *Information Processing & Management*, page 102054, 2019.

L. Kretz. Virtual online community with geographically targeted advertising, Apr. 17 2008. US Patent App. 11/580,725.

T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *SIGKDD*, 2009.

C.-T. Li and M.-K. Shan. Team formation for generalized tasks in expertise social networks. In *SocialCom*, pages 9–16, 2010.

J. Li, X. Wang, K. Deng, X. Yang, T. Sellis, and J. X. Yu. Most influential community search over large social networks. In *ICDE*, pages 871–882, 2017.

R.-H. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Skyline community search in multi-valued networks. In *SIGMOD*, pages 457–472. ACM, 2018.

R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5):509–520, 2015.

R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai. Persistent community search in temporal networks. In *ICDE*, pages 797–808. IEEE, 2018.

Y. Liu, A. Niculescu-Mizil, and W. Gryc. Topic-link lda: joint models of topic and author community. In *ICML*, pages 665–672, 2009.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

L. Muflikhah and B. Baharudin. Document clustering using concept space and cosine similarity measurement. In *2009 International Conference on Computer Technology and Development*, volume 1, pages 58–62. IEEE, 2009.

J. Naruchitparames, M. H. Güneş, and S. J. Louis. Friend recommendations in social networks using genetic algorithms and network topology. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2207–2214. IEEE, 2011.

J. Perkins. *Python 3 text processing with NLTK 3 cookbook*. Packt Publishing Ltd, 2014.

Y. Ruan, D. Fuhry, and S. Parthasarathy. Efficient community detection in large networks using content and links. In *WWW*, pages 1089–1098, 2013.

M. Sachan, D. Contractor, T. A. Faruquie, and L. V. Subramaniam. Using content and interactions for discovering communities in social networks. In *WWW*, pages 331–340. ACM, 2012.

N. Seco, T. Veale, and J. Hayes. An intrinsic information content metric for semantic similarity in wordnet. In *Ecai*, volume 16, page 1089, 2004.

M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*, pages 939–948, 2010.

J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*, pages 990–998, 2008.

J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *ICDE*, pages 233–244. IEEE, 2018.

Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on Learning Theory*, pages 25–54, 2013.

Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516. ACM, 2012.

D.-N. Yang, Y.-L. Chen, W.-C. Lee, and M.-S. Chen. On social-temporal group query with acquaintance constraint. *PVLDB*, 4(6):397–408, 2011.

L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang. Index-based densest clique percolation community search in networks. *IEEE TKDE*, 30(5):922–935, 2017.

Z. Zhang, X. Huang, J. Xu, B. Choi, and Z. Shang. Keyword-centric community search. In *ICDE*, pages 422–433. IEEE, 2019.

D. Zheng, J. Liu, R.-H. Li, C. Aslay, Y.-C. Chen, and X. Huang. Querying intimate-core groups in weighted graphs. In *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*, pages 156–163. IEEE, 2017.

Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

Q. Zhu, H. Hu, C. Xu, J. Xu, and W.-C. Lee. Geo-social group queries with minimum acquaintance constraints. *The VLDB Journal*, 26(5):709–727, 2017.